New Results and Insighs on ForkAE

Elena Andreeva[1], Arne Deprez[2], Jowan Pittevils[2],
Arnab Roy[1], Amit Singh Bhati[2], and Damian Vizár[5]

[1] AAU Klagenfurt, Austria
elena.andreeva@aau.at, arnab.roy@aau.at
[2] imec-COSIC, KU Leuven, Belgium
arne.deprez1@gmail.com, amitsingh.bhati@esat.kuleuven.be,
jowan.pittevils@student.kuleuven.be
[3] CSEM, Switzerland
damian.vizar@csem.ch

Abstract. In this work we summarize, discuss and interpret the recent advances in security analysis and optimized SW and HW implementations of the NIST second round lightweight authenticated encryption candidate ForkAE. We highlight the most important comparisons in the recent ForkAE results, discuss their conclusions and illustrate our interpretation of these results with further use cases.
While ForkAE with the ForkSkinny primitive is designed with efficiency for short messages in mind, when the underlying primitive ForkSkinny is used in the recent modes of operation RPAEF and GCTR the performance is also improved for the longer messages for authenticated encryption and encryption, respectively.
Regarding security, we bring evidence that ForkAE provides stronger security guarantees than originally claimed, and can thus also be classified as a '*defense-in-depth*' candidate. More precisely, ForkAE provides well-defined nonce misuse resistance MNR guarantees in its sequential SAEF mode of operation (without any modifications) which is a feature that ensures stronger security guarantees in scenarios where nonces may accidentally or maliciously be forced to repeat. The same security property of the SAEF mode simultaneously offers additional protections against blockwise adversaries, a use case particularly relevant in the lightweight setting. Furthermore, both parallel modes PAEF and the recent RPAEF come with full n-bit security. In addition, we show that when the ForkSkinny primitive is used in a GCTR counter-mode style encryption – a use case relevant when no explicit authentication is mandated, the resulting schemes come not only with good performance for longer messages but also with high security guarantees.
With respect to SW and HW implementation results we highlight the recent ForkAE and ForkSkinny primitive optimized SW and HW implementation strategies and their comparisons with other second round candidates, such as the SKINNY-AEAD and Romulus designs. The referenced optimized SW implementation comes with important improvements over the existing ForkAE portable implementations of Rhys Weatherley, with table-based and parallel optimization techniques.

1 Introduction

ForkAE is a family of cryptographic algorithms for lightweight authenticated encryption and a second-round candidate in the NIST Lightweight cryptography standardization process. Its primary target use case are applications of predominantly short message size (e.g., 8, 16, 32 bytes). This class of applications covers an ever increasing range of practical scenarios, such as: the Secure Onboard Communication in the automotive industry [4] which are expected to handle short messages with stringent latency requirements; Critical communication and massive IoT domains of 5G where frequent bursts of very short messages [1] need to proccessed; Narrowband IoT which allows for a minimum payload

size of 16 bits [3, 2], and which will dominate the communication in applications such as smart parking lots that need to transmit information encoded on a few bits (e.g., "free" or "occupied" status); medical implant devices, such as pacemakers, transmit messages of length at most 16 bytes to and from the device programmer; Advanced robotic prosthetics which wirelessly transmit bursts of short messages with stringent latency requirements, as well as 1-byte temporal synchronization messages [5]; Wireless aircraft tyre pressure monitoring systems which usually transmit payloads of ≤ 10 bytes [28], etc.

Most recently we have witnessed several works investigating various security and implementation aspects of ForkAE. In this work we give an overview of these results and discuss their contributions to the advancement of knowledge on the ForkAE lightweight authenticated encryption design and their implications on the use of ForkAE. Our survey covers implementation-oriented results both in hardware and software, and as well as security results both on the level of primitive and mode. We also point to further target use cases for ForkAE, such as *defense in depth* and much more efficient long message processing.

Finally, we **we identify new applications for the forkcipher primitive, as well as promising future research directions**.

2 Brief Description

A forkcipher is a function $\mathsf{F} : \{0,1\}^\kappa \times \{0,1\}^t \times \{0,1\}^n \times \{0,1,\mathsf{b}\} \rightarrow \{0,1\}^n \cup \{0,1\}^{2n}$ which takes a tweakey $\in \{0,1\}^{t+\kappa}$, a message $\in \{0,1\}^n$ as and an output-switch as input and produces the "left", the "right" or "both" n-bit output blocks according to the output-switch. κ and t denotes the length (in bits) of the secret key and tweak respectively.

ForkSkinny is a forkcipher function and the underlying primitive used in the ForkAE NIST submission [9]. It is constructed following the iterate-fork-iterate (IFI) paradigm using the tweakable block cipher Skinny[14]. The outline of the ForkSkinny construction is depicted as in Figure 1.

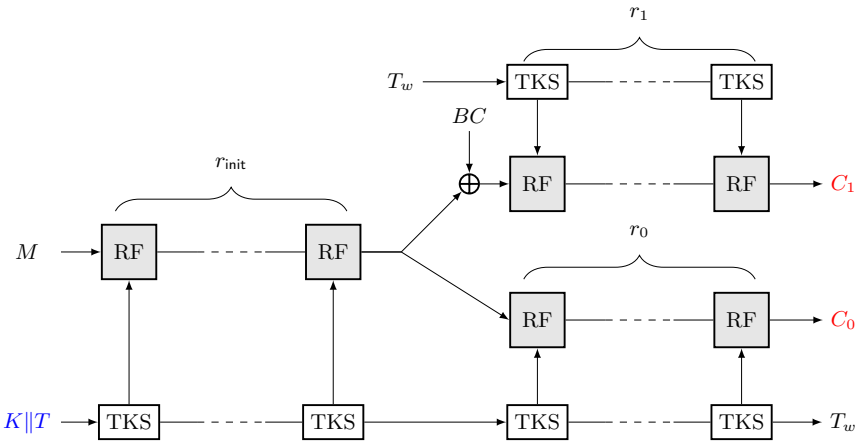


Fig. 1: The structure of ForkSkinny. TKS denote the round tweakey schedule function and RF denotes the round function.(output-switch omitted)

The round function of ForkSkinny is almost identical to the round function of Skinny. Each round can be described as

$$\mathcal{R}_i = \texttt{Mixcolumn} \circ \texttt{Addconstants} \circ \texttt{Addroundtweakey} \circ \texttt{Shiftrow} \circ \texttt{Subcell}$$

where the `Mixcolumn`, `Shiftrow`, `Subcell` and `Addroundtweakey` functions are same as in Skinny. Note that the `Addroundtweakey` function is used in ForkSkinny to generate round tweakeys for $r_{\mathsf{init}} + r_0 + r_1$ rounds, where r_0 and r_1 denote the number of rounds in the left and right branch of ForkSkinny and r_{init} denotes the number of rounds before forking. The `Addconstants` function in ForkSkinny differs from Skinny. Unlike Skinny (which has 6 bit round constants), the `Addconstants` in ForkSkinny generates 7 bit round constants using an LFSR. For a more detailed description of the ForkSkinny algorithm we refer the readers to the article [9].

ForkAE proposed two modes of operation with ForkSkinny – a sequential one SAEF and a parallel one PAEF.

3 Security of ForkSkinny and ForkAE

3.1 Cryptanalysis of Forkcipher: An Overview

In general, in an instantiaion of a forkcipher any exploitable weakness in the forking structure can give advanatge to an adversary. For example, on an earlier instantiation of forkcipher - ForkAES, the recosntruction query was used to cryptanalyse [11] and mount an attack on 9 out of 10 rounds. The cryptanalysis of ForkAES [11] exploited the (tweakey) differential in AES particularly in the forking setup. This differential trail is reffered to as reflection trail. Although the cryptanalysis in [11] did not violate the security of the ForkAES, it certainly showed the possible weaknesses in the forking structure. It also reduced the security margin of ForkAES significantly in terms of number of rounds. In [12], Bariant et al. improved upon the cryptanalysis of ForkAES proposed in [11]. They proposed an attack on the full ForkAES and showed several theoretical attacks on the full ForkAES by identifying a fraction of weak keys.

ForkSkinny does not contain the exploitable weakness used against ForkAES. In [12], used a different approach to cryptanalyse reduced round ForkSkinny. The paper provided cryptanalysis of ForkSkinny-128 with 256-bit key and 256-bit tweakey (128-bit key and 128-bit tweak).

3.2 Security of ForkSkinny

Security of ForkSkinny depends heavily on the security of the block cipher SKINNY. It is clear that any cryptanalysis on (reduced round) SKINNY can also be applied to ForkSkinny. The known cryptanalysis of reduced round ForkSkinny does *not* cover significantly more number of rounds than SKINNY. For example, the most recent third party cryptanalysis [12] of ForkSkinny-128 with 256-bit key and 256-bit tweakey (128-bit key and 128-bit tweak) covers 26 and 24 rounds respectively. This is not far off from the best known attack on SKINNY, which is on 23 rounds. The cryptanalysis in [12]

does not contradict, as also concurred by the authors of the paper, the security of ForkSkinny and ForkAE. Here we remark that ForkSkinny-128 with 256-bit key is not a valid instantiation according to the ForkSkinny specification submitted to the NIST.

Overall, the known cryptanalysis results on ForkSkinny (and on forkcipher) confirms the designers perspective that ForkSkinny has a security margin that is comparable to SKINNY.

3.3 New Security Results on SAEF

Several instances of the SAEF mode for nonce-based AEAD have been submitted as part of the ForkAE algorithm family [9]. SAEF is a sequential design, primarily conceived to have a reduced memory footprint compared to the PAEF mode. The original security claim given for SAEF was $n/2$-bit nonce-based AEAD security. In a recent work, Andreeva et al. [8] show that SAEF is in fact an a secure OAE scheme. We give the necessary context, the formal statement of the new result, and discuss the implications.

Online Authenticated Encryption Security Notion. The security notion of Online Authenticated Encryption (OAE) is one of the security notions that aim at mitigating the impact accidental nonce reuse on AEAD (a.k.a. *nonce misuse*). Even infrequent nonce-reuse can have catastrophic consequences on the security of AE schemes targeting the basic nonce-based AEAD security [35]. The severity and plausibility of nonce reuse in practice has been demonstrated by real-world attacks. For example, Böck et al. in USENIX'16 [15] demonstrated that 184 HTTPS servers worldwide repeat nonces used with AES-GCM, leading to a complete break of authenticity in the TLS session. Vanhoef and Piesens at CCS'17 [34] presented an attack which forces nonce repetitions and breaks the WPA2 wireless protocol. The risk of an accidental nonce reuse is also high for small devices, such as in IoT, due to constraints on memory, energy etc.

The "best possible" security of nonce based AEAD against nonce misuse captured by the notions of MRAE [31] and the RAE [21] is, unfortunately, mutually exclusive with the AE scheme in question being online. An online AE scheme processes the plaintext as in smaller, typically fixed-size blocks during encryption, and computes the ciphertext as a sequence of such blocks, such that i-th ciphertext block can be computed after having seen the first i plaintext blocks only. Practical AE schemes that are online can typically also be implemented with a constant memory footprint.

¡¡¡¡¡¡¡ HEAD The security notion called OAE proposed by Fleischmann et al. [18] (and later corrected by Hoang et al. [21]) slots between the notions of nonce-based AEAD and MRAE, capturing the same level of integrity (as MRAE) and a well-defined, albeit lower level of confidentiality in face of nonce misuse, which is achievable by online AE schemes. ======= The security notion called OAE proposed by Fleischmann et al. [18] (and later corrected by Hoang et al. [22]) slots between the notions of nonce-based AEAD and MRAE, capturing the same level of integrity (as MRAE) and a well-defined, albeit lower level of confidentiality in face of nonce misuse, which is achievable by online AE schemes. ¿¿¿¿¿¿¿ 67fe6bdbc01d176f2633984e5bc96f4aab09d87a Roughly speaking, for an underlying primitive's blocksize, an OAE-secure scheme will leak the length of longest block-aligned prefix of two plaintexts encrypted with the same nonce and associated data but nothing more. Endignoux and Vizár [20] later showed that the equivalence of OAE-security to (an

adapted version of) the blockwise-adaptive AE security from [19]. The result implies that OAE-secure schemes are safe to use in settings where block-wise adaptive attackers may exist, i.e., where an application outputs a part of the ciphertext before it has been fed the entire plaintext. Such attacks pose a real threat to lightweight applications, e.g., due to small devices not equipped with sufficient memory. The OAE notion has been targeted by the COLM [6] CAESAR defense in depth finalist.

OAE security of SAEF.

In [8] the authors investigate the OAE security of ForkAE and more precisely on the online scheme SAEF. ForkAE was proposed with two modes initially in mind: the parallelizable mode PAEF and the sequential mode SAEF [9]. Both are provably secure nonce-based AE schemes [30] (we skip the syntax of AE schemes for brevity). The former achieves optimal quantitative security (thus allowing for secure instances with a small block size) while the latter is secure up to the birthday bound but requires a smaller internal state. We focus on the SAEF mode which processes blocks of AD and message with single call to F each, using tweaks composed of the either a padded nonce (of length $t-4$) or a string of $n-3$ zeros, and a domain separation constant. See Figure 2.

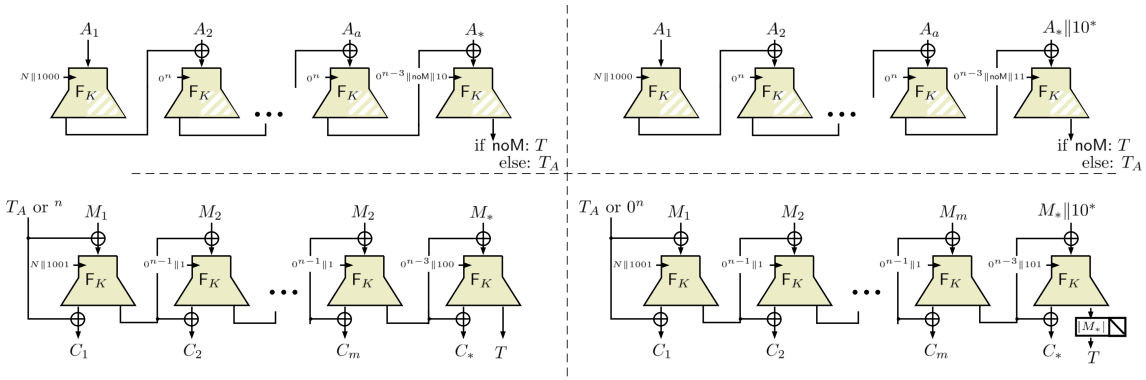


Fig. 2: The encryption algorithm of SAEF[F] mode. The bit $\mathsf{noM} = 1$ iff $|M| = 0$. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed. If $|M| = 0$, the tag is equal to T_A; else if $|A| = 0$, the AD processing is skipped.

The results of [8], stated below in Theorem 1, show that the SAEF mode is provably OAE secure *without the need of applying any design modifications*, as long as the total amount of data processed with a single key is $\ll 2^{n/2}$ blocks, with n the blocksize of the underling ForkSkinny cipher. This implies that SAEF guarantees qualitatively stronger security than claimed in the original submission at unchanged quantitative levels. With the efficiency gain due to ForkSkinny, SAEF is conjectured to outperform COLM instantiated with the SKINNY tweakable blockcipher (with the same parameters).

Theorem 1 ([8]). *Let F be a tweakable forkcipher with n-bit block. Then for any nonce-misuse adversary $\mathcal{A}$ who makes at most $q_e \leq 2^{n-1}$ encryption queries, at most q_v decryption queries such that the total number of forkcipher calls induced by all the queries is at*

most σ, we have

$$\mathbf{Adv}^{\mathbf{oprpf}}_{\mathrm{SAEF[F]}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{B}) + \frac{3 \cdot \sigma^2}{2^{n+1}}$$
$$\mathbf{Adv}^{\mathbf{auth}}_{\mathrm{SAEF[F]}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathbf{prtfp}}_{\mathsf{F}}(\mathcal{C}) + \frac{\sigma^2 + 4 \cdot q_v}{2^n}$$

for some adversaries $\mathcal{B}$ and $\mathcal{C}$, each making at most 2σ queries, and running in time given by the running time of $\mathcal{A}$ plus $\gamma \cdot \sigma$ for some "small" constant γ.

Use Cases. The newly-proved security properties of SAEF will greatly benefit lightweight applications (such as consumer grade and low-power IoT), where it is often necessary to compute ciphertext blocks *on-the-fly* with constant memory and/or latency. Devices used in the said applications additionally come with stringent cost constraints and often get inadequate security as a consequence [25, 24], and would thus greatly benefit from a lightweight AEAD scheme with robust security. We exemplify how an OAE secure scheme can be useful in the context of lightweight cryptography applications.

Nonce reuse in lightweight applications. Many lightweight applications have stringent cost constraints on the embedded platforms they use. The pertinent platforms thus cannot benefit from most robust HW, while at the same time, these HW platforms may realistically be exposed to physical attackers in many applications (outdoor smart-home equipment accessible from the street, smart city infrastructure being exposed by its very nature, intelligent building sensors and actuators being accessible from maintenance access ports etc). As a consequence, certain sources of accidental nonce reuse (such as a reset occurring during counter incrementation, lack of persistent storage to store nonce value between resets etc) may be exploited, or even artificially amplified by an attacker with physical access. Having an AE scheme that is both sufficiently lightweight and maintaining a well-defined level of security in face of nonce-reuse is truly valuable in such a cases.

Nonce reuse and short messages. Informally speaking, the schemes secure in the sense of the MRAE notion [31] will retain the same security as nonce-based AEAD, even if nonces repeat arbitrarily, except for unavoidably leaking the information that the complete plaintext tuple (N, A, M) repeats (because this implies repetition of ciphertexts). For encryption queries with a single-block plaintext (up to 8 or 16 bytes with the algorithms in the ForkAE family), the security afforded by OAE schemes is equivalent with that of MRAE schemes, as the leakage of the length of longest common blockwise prefix is the same as leakage of plaintext equality. The main use case targeted by ForkAE are applications where the communication is dominated by (very) short messages. An application would thus get the "best possible" nonce-misuse protection for a majority of messages while benefiting from the efficiency of an OAE scheme.

OAE-aware engineering. In 2015, Hoang et al. warned that OAE confidentiality guarantees are not as strong as originally believed [22]. They presented the CPSS (chosen prefix secret suffix) attack, which recovers a valuable secret string that is placed in many OAE encryption queries of the same session. More precisely, if sufficiently many plaintexts are composed by appending such a secret string S to an attacker-controlled variable-length prefix P with a small enough granularity b (typically $b = 8$ for a byte), then the attacker can recover S in a chosen plaintext attack with a query complexity $\lceil |S|/b \rceil * 2^b$ by using the prefix-preservation property to exhaustively search for the value of one byte of S after another. Hoang et al. gave the HTTP session cookie as an example of such a setting; in

the context of lightweight-crypto applications, an example of such a repeatedly used secret string are device authentication passwords, such as used for MQTT [33].

While OAE falls prey to CPSS attack *in general*, its confidentiality guarantees are still sufficient in many applications, if sufficient attention is given to the design of the the higher-level security layer using the OAE scheme. To counter CPSS, it is sufficient to ensure that a repeatedly used high-value secret is only preceded by a *fixed-size* prefix. This is also the case for the password field in MQTT's CONNECT packet, provided the implementation does not allow the clientID and userName fields to be changed arbitrarily.

Blockwise encryption for external flash. Applications under the umbrella of "Internet of Things" typically comprise connected devices based on low-cost embedded platforms (the "Things"). External flash memory is used in many such applications to extend the storage capacity of such platforms; the advantage is its low cost, the downside is that it is extremely easily accessible to an attacker with a physical access to the device. Even if care is taken not to store any sensitive data on the external flash, this is unavoidable in some cases. For example, a new firmware image received during a remote firmware update in a micro-controller may not fit into the internal flash memory alongside the currently running image, and so is routinely stored in the external flash until the update is complete. In order to protect valuable intellectual property, as well as prevent malicious firmware modifications, the firmware image must be encrypted an authenticated.

In a typical configuration (such as in the popular Nordic NRF52840 [32] to give an example) the received firmware image is highly unlikely to fit into the RAM, which means that the micro-controller has to compute the ciphertext on the fly and write first ciphertext blocks to the external flash memory, possibly before the last blocks of the new image have been received. An OAE secure scheme is a very useful tool in this case, because it allows exactly this kind of on-the-fly processing, and offers provable security guarantees against (physical) attackers that may tamper with the firmware image adaptively, while observing the ciphertext blocks already written.

This observation generalizes to any other large amounts of data that need to be stored on the external flash and require authenticated encryption.

3.4 n-bit Secure RPAEF for Longer Messages

In addition to SAEF and PAEF, in [10] Andreeva et al. also propose a new authenticated encryption mode called RPAEF (for Reduced PAEF). RPAEF (ee Figure 3.) can be seen as the mode ΘCB with a new, forkcipher based finalization.

Namely, it uses a single branch of the ForkSkinny primitive for all but the finalization call where both branches of ForkSkinny are evaluated. The authentication in RPAEF is enabled by an aggregate message blocks sum which is fed into the final ForkSkinny tweak. In terms of number of executed primitive rounds, RPAEF is *always* (for all message lengths) more efficient than the respective tweakable blockcipher modes.

When instantiated with ForkSkinny, RPAEF computes the equivalent of $\ell - 1$ calls to SKINNY and 1 call to ForkSkinny for a message of ℓ blocks as compared to at least $\ell+1$ SKINNY calls in the most optimal ΘCB tweakable cipher mode. This general RPAEF

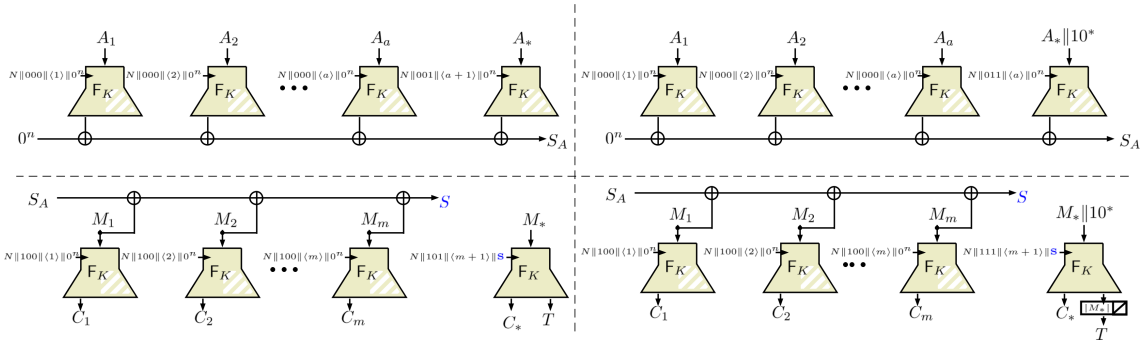


Fig. 3: The encryption algorithm of RPAEF[F] mode. The picture illustrates the processing of AD when length of AD is a multiple of n (**top left**) and when the length of AD is not a multiple of n (**top right**), and the processing of the message when length of the message is a multiple of n (**bottom left**) and when the length of message is not a multiple of n (**bottom right**). The white hatching denotes that an output block is not computed. If $|M| = 0$, the tag is equal to T_A; else if $|A| = 0$, the AD processing is skipped.

mode optimization comes at the cost of using of large tweaks (as large as 256 bits) in the final ForkSkinny call and increased HW area footprint.

Security-wise, a similar proof to the proof of the PAEF construction applies to RPAEF and this is formally proven in Section 6.7, Theorem 3, [10] showing that RPAEF achieves optimal n-bit quantitative security.

Implications. RPAEF is the second parallel AE mode (after PAEF) that offers full n-bit security and in the spirit of the past CAESAR AE competition places ForkAE in a category of lightweight authenticated encryption schemes with high security margin and hence the 'defense in depth' category. From a practical perspective, this enables use cases where the key will never be refreshed and enables life-long device utilization span.

3.5 Counter Mode Beyond Birthday Secure Encryption

In [7] the authors introduce a *generic* CTR encryption mode GCTR that uses an multi-forkcipher (and forkciphers in particular) as an underlying primitive on inputs a message M, a nonce N, a block counter j (as the conventional nonce-based CTR mode), and an additional random IV. The work gives the security analysis of 36 "simple and natural" GCTR variants and an additional GCTR-X slightly more involved variant under the nivE security notion by Peyrin and Seurin from [26]. The authors identify numerous security and efficiency advantages and tradeoffs of these modes both in general and more concretely, when the modes are instantiated with ForkSkinny.

The results show that many of the schemes achieve from well beyond birthday BB to *full* n-bit security under nonce respecting adversaries and some even beyond birthday BB and close to *full* n-bit security in the face of realistic nonce misuse conditions. Most variants' security bounds are better than that of classical CTR mode, which becomes void at $\approx 2^{n/2}$ processed blocks.

The authors give a comparison of GCTR using ForkSkinny (a multi-forkcipher MFC with two branches $s = 2$) with the traditional CTR and the more recent CTRT modes where both are instantiated with the SKINNY tweakable cipher. The estimations show that GCTR with ForkSkinny can achieve an efficiency advantage over 20% for the

longer messages. The efficiency comparison between GCTR, CTRT and basic CTR in [7], Figure 4 is done by comparing the total number of primitive rounds for instances based on ForkSkinny [29]. The figure illustrates that the number of rounds required for a GCTR_2-ForkSkinny is smaller than CTR-SKINNY and CTRT-SKINNY for all values of queried bytes. In fact, any GCTR_2-ForkSkinny mode with $t = 2n$ is still more efficient than the CTRT-SKINNY mode with $t = n$.

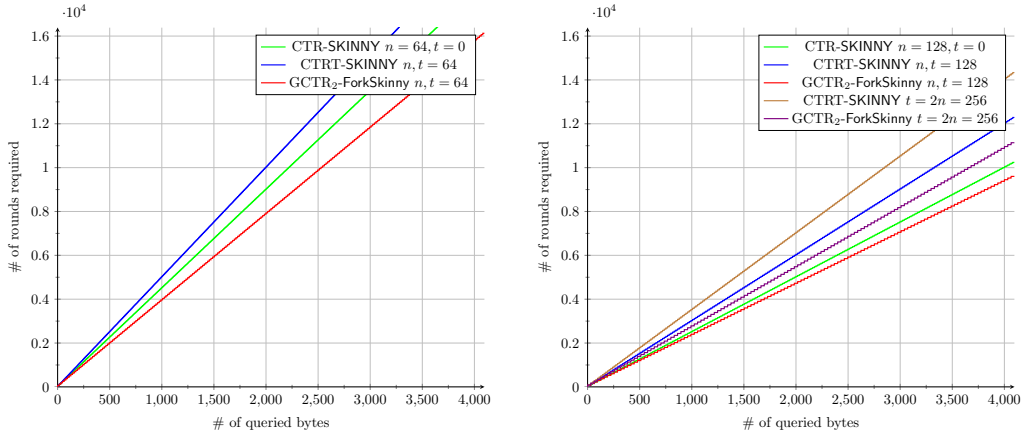


Fig. 4: Efficiency comparison of any GCTR_2 mode (instantiated with ForkSkinny) with CTR and CTRT modes (instantiated with SKINNY). These plots show the number of rounds required to process the queried bytes ($sn\sigma/8$). The left figure corresponds to the input size $n = 64$ bits whereas the right figure corresponds to the input size $n = 128$ bits.

Implications. In lightweight scenarios where no explicit authentication is required or when a lightweight encryption scheme is used for randomness or subkey generation ForkSkinny in a GCTR mode can in a very efficient way offer the highest n-bit securiy both in the nonce misuse and nonce respecting scenarios. This brings in an additional aspect of the 'defense in depth' power of the ForkSkinny primitive when used in an counter-style encryption.

4 SW and HW Implementation Results

In ASIACRYPT 2019 Andreeve et al. [10] published the first work (up to our knowledge) that shows optimizied ForkAE HW implementations and performance comparisons (including the RPAEF mode). More concretely, Section 7, Table 10 of this work provides a hardware comparison of the three modes PAEF, SAEF and RPAEF (instantiated with different ForkSkinny variants) with Sk-AEAD. The Sk-AEAD is the tweakable cipher mode TAE [23] instantiated with Skinny-AEAD M1/M2, M5/M6 [13]. Based on the round-based implementations all these three modes are shown to perform faster (in terms of cycles) for short data (up to 3 blocks) with about the same area. RPAEF beats its competitor for all message sizes at the cost of a area increase of about 20% (for only one of its variants). The performances are further optimized by exploiting the in-built parallelism (//)in the ForkSkinny primitive. This allows for further improved performance results with that strategy to be obtained. For messages up to three 128-bit blocks, the speed-up of PAEF and SAEF (both parallel(//)) ranges from 25% to 50%, where the advantage is largest for

the single-block messages. Most importantly, the RPEAF, PAEF, and SAEF(//)instances result in fewer cycles than the θCB variants for all message sizes at a small cost in area increase. However, the relative advantage of the latter instances is more explicit for short messages; as it diminishes asymptotically with the message blocks. For message sizes up to 8 bytes the PAEF-ForkSkinny-64-192 instances are more than 58% faster with also a considerably smaller implementation size. We refer the reader for detailed read to Section 7, Table 10 of [10]. Building on the latter work Purnal et al. [29] worked on optimized HW implementations of ForkAE in the First NIST 2019 workshop.

4.1 Improved HW Implementation Strategies

In his recent master thesis [27] on the topic also studies HW optimizations with a main focus on area and throughput efficiencies. The thesis offers two new (Restart and Retrace) area-focused architectures which are compared with a balanced (Forkreg) and a throughput-focused (Parallel) architecture. The work focuses on primitive-level optimizations regarding the the ForkSkinny cipher. Overall, the work explores 4 architectures:

- Parallel: This implementation calculates the two ForkSkinny branches simultaneously in parallel. When looking at area-effciency, this design is worse than a serial implementation. However, it does not take up double as much area as one might assume when thinking of an implementation with dual parallelism. For small messages, a round-based encryption-only Parallel ForkSkinny implementation can have higher throughput and area than the round-based encryption-only SKINNY [4].
- Forkreg: an extra block-sized register called the fork register is used to store the fork-state and to read this state once the ciphertext part C_0 has been calculated.
- Restart: This architecture restarts from the beginning once C_0 is ready to compute the other branch. The main drawback is that the plaintext-data must remain available for at least $r_{init} + r_0$ rounds.
- Retrace: This architecture is more flexible than Restart. It allows for both encryption and decryption and the plaintext can be forgotten once the internal state has been loaded. This is accomplished by, once C_0 has been calculated, decrypting the internal state back up to the point of the fork and then re-encrypting it to become C_1.

The latter three area-efficient architectures (excluding the parallel) are compared to a SKINNY implementation (see Table 5) where the results were obtained using the NanGate 45NM library for ASICS under normal operating conditions, and where no clock gating or latches are used, as well as a similar datapath sizes of one sixteenth of the block size are used. For the ForkSkinny implementations, the throughput is defined as a range instead of a fixed number. This is to account for the fact that for very short messages, the throughput is effectively doubled if both outputs of the forkcipher are used. As such, the range is defined as worst case (very long messages) to best case (very short messages). An important result to note is that the Restart ForkSkinny-128-288 implementation uses less area than the Skinny-128-384 implementation. This is due to the fact that both ForkSkinny-128-192 and ForkSkinny-128-288 only need a half block-sized register to store the final block of the tweakey state, whereas Skinny-128-384 uses a full block-sized register to store the final part of the tweakey state.

These results show that the in the Retrace architecture, the possibility to decrypt comes nearly for free in terms of area – there is only a very small area-difference between the

encryption-only and encryption/decryption implementation. As the number of cycles for a decryption of the ForkReg and Retrace architecture are the same, an area-efficient design that must mainly decrypt messages and only rarely encrypt them should probably use the Retrace architecture. A significant difference between encryption-only and encryption/decryption Retrace architecture is the critical path, which does change by a large amount when going from encryption-only to encryption/decryption, especially for the 128-288 architecture. The encryption/decryption design of Retrace can have higher throughput when decrypting messages (for small messages) than encryption/decryption SKINNY. This is mainly due to the fact that decrypting requires the key to be updated before the real decryption can begin, which nearly doubles the amount of cycles for SKINNY and Forkreg ForkSkinny but the increase in cycles is (relatively speaking) less pronounced for Retrace ForkSkinny. The area penalty of forking in an encryption/decryption implementation of ForkSkinny can be hidden almost entirely at the cost of cycles necessary to process message blocks.

Algorithm	Architecture	Area [GE]	Area [GE]	Delay [ns]	Delay [ns]	Cycles	Cycles	T'put* [kb/s]	T'put* [kb/s]
		E-only	EncDec	E-only	EncDec	Enc	Dec	E-only	EncDec
Skinny-64-192		2448	3754	1.04	1.62	872	1698	7.33	3.7
Skinny-128-256		3525	5499	0.94	1.67	1040	2034	12.29	6.2
Skinny-128-384		4680	7157	0.93	1.74	1208	2370	10.58	5.4
ForkSkinny-64-192	Restart	**2718**	na	1.15	na	2218	na	2.8-5.7	na
ForkSkinny-128-256	Restart	**3917**	na	1.19	na	2638	na	4.8-9.7	na
ForkSkinny-128-288	Restart	**4567**	na	1.39	na	3058	na	4.2-8.4	na
ForkSkinny-64-192	Retrace	3867	**3894**	1.67	1.72	2328	2770	2.7-5.5	2.3-**4.6**
ForkSkinny-128-256	Retrace	5648	**5685**	1.73	1.81	2748	3298	4.7-9.3	3.9-**7.8**
ForkSkinny-128-288	Retrace	6645	**6650**	1.92	2.23	3168	3826	4.0-8.1	3.3-**6.7**
ForkSkinny-64-192	ForkReg	3243	4470	0.93	1.90	1362	2770	4.7-**9.3**	2.3-**4.6**
ForkSkinny-128-256	ForkReg	4977	6787	1.4	2.02	1614	3298	7.9-**15.9**	3.9-**7.8**
ForkSkinny-128-288	ForkReg	5629	7795	1.23	2.11	1866	3168	6.8-**13.7**	3.3-**6.7**

(SFF/MUX/XOR/NAND with 7.67/2.33/2/1 GE)
* Throughput at 100 kHz

Fig. 5: Results for the different word-based implementations

4.2 Improved SW Implementation

The main contribution of the work of [16, 17] is the development of an optimized software implementations for ForkAE.

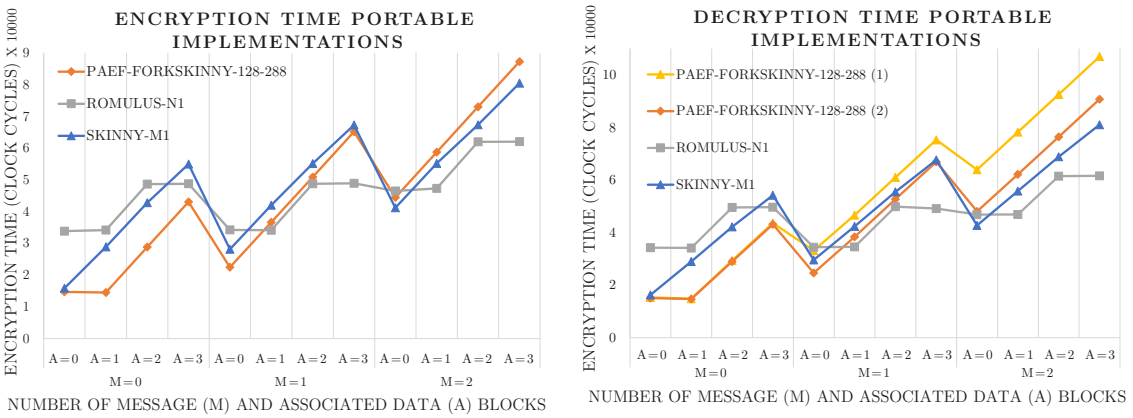
The authors analyze the performance and efficiency of different ForkAE implementations on two embedded platforms: ARM Cortex-A9 and ARM Cortex-M0.

The first improvement deals with the portable ForkAE implementations. A decryption optimization technique is applied which allows to accelerate the existing portable implemetation of [36] for decryption by up to 35%.

Secondly, platform-specific software optimizations are explored. In platforms where cache-timing attacks are not a risk, the authors present an S-box implementation which improves the performance of the ForkSkinny round function and contributed to a more efficient desing for table based implementations. More generally, on such platforms the SKINNY round function and inverse round function can be transformed into a combination of table-lookups which allows to for a significant increase in performance. Compared to the existing portable implementations, the latter technique speeds up both encryption and decryption by 20% and 25%, respectively. The impact on the amount of memory needed for this implementation can be minimised by reducing the number of tables.

Furthermore, [16, 17] propose a set of platform-specific optimizations for processors with parallel hardware extensions such as ARM Neon. Without the need of relying on parallelism provided by long messages (bitsliced implementations), the authors focus on the data-level parallelism in ForkSkinny. The latter can be used to increase efficiency and reduce latency for the small messages that are typical in IoT applications and are the target use case for the ForkAE algorithm.

Finally, the performance of the implementations on the ARM Cortex-M0 and ARM Cortex-A9 processors are benchmarked and a comparison with the other SKINNY-based schemes in the NIST lightweight competition – SKINNY-AEAD and Romulus is given. We elaborate on the performance comparison from this work of ForkAE with other SKINNY-based AEAD schemes Romulus and SKINNY-AEAD in in Figure 6a and Figure 6b from [17]. The primary instances of the NIST LWC submission for small messages with different number of message (M) and associated data (A) blocks are compared. These figures highlight the advantage of a forkcipher over a blockcipher for encryption of small messages.



(a) (b)

Fig. 6: Performance comparison of SKINNY based ciphers on Cortex-A9 from [17]. Encryption with implementations from [36]. Decryption with SKINNY, Romulus and PAEF-ForkSkinny-128-288(1) implementations from [36] and PAEF-ForkSkinny-128-288(2) implementation with preprocessed tweakey schedule.

Table-based Implementations The lookup table round function is implemented on the STM32F0 platform with the ARM Cortex-M0 processor as an example lightweight platform with no cache. Then, the different trade-offs between speed, code size and memory usage are explored. Table 1 lists the results for an implementation with 4 different lookup tables. Compared to the portable implementations of [36], it needs up to 16% fewer clock cycles when the tables are stored in ROM. When the tables are stored in RAM, this gain is almost 20%.

ForkSkinny gains here in speed in exchange for a higher memory cost. Particularly, when four lookup tables are used in combination with two tables containing the mixed and shifted round constants, a total of 4.7 kB of memory is needed. We show that the impact of the lookup table implementation on the memory usage can be greatly reduced when only one T-table is used instead four. The performance results for this method are

listed in Table 1. This approach introduces some extra calculations in the round function, but as can be seen from the results, the impact on the computation time is only a few cycles per byte. The reduction in memory cost of 3 kB is significant and can be very important for the resource-constrained devices in embedded applications.

Table 1: Implementation figures for the table-based ForkAE encryption implementation on the ARM Cortex-M0 from [17].

Encryption	**Tables in ROM**			**Tables in RAM**		
4 lookup tables	c/B	ROM	RAM	c/B	ROM	RAM
PAEF-FS-128-192	2110	6752	192	2016	1960	4984
PAEF-FS-128-256	2111	6748	200	2017	1956	4992
PAEF-FS-128-288	2859	7034	220	2739	2242	5012
SAEF-FS-128-192	2128	6688	192	2035	1896	4984
SAEF-FS-128-256	2129	6674	200	2035	1882	4992
1 lookup table						
PAEF-FS-128-192	2138	3692	192	2030	1972	1912
PAEF-FS-128-256	2139	3688	200	2031	1968	1920
PAEF-FS-128-288	2919	3980	220	2805	2260	1940
SAEF-FS-128-192	2157	3628	192	2049	1908	1912
SAEF-FS-128-256	2157	3614	200	2049	1894	1920

To study the gain of tabulating the inverse round function, the performance of one specific implementation for table-based decryption is analysed. The implementation uses one lookup-table that is stored in ROM and a preprocessed tweakey schedule that is stored in RAM. The performance metrics are listed in Table 2. When this is compared with the portable decryption implementation, it can be seen that using lookup tables can significantly speed-up decryption, as the amount of cycles that are needed is reduced with up to 25%. This speed-up is higher than for encryption because of the simpler inverse round function where the addition of the round tweakey and constants can be done at the end.

Table 2: Implementation figures for the table-based ForkAE decryption implementation on the ARM Cortex-M0.

Decryption	c/B	ROM	RAM
PAEF-FS-128-192	2241	3261	818
PAEF-FS-128-256	2241	3253	826
PAEF-FS-128-288	3156	3529	958
SAEF-FS-128-192	2259	3317	818
SAEF-FS-128-256	2257	3303	826

Parallel Implementations In Table 3 the performance of ForkAE encryption and decryption on the ZYBO platform is illustrated when the Neon SIMD implementations are used. For ForkSkinny instances with a block-size $n = 128$, the S-box and its inverse are replaced with the 128-bit Neon implementation. For `PAEF-ForkSkinny-64-192`, the ForkSkinny implementation with parallel round function is used for encryption. Its decryption only features the parallel S-box layer, as it cannot benefit from parallelism in the round function.

For the 128-bit instances with the Neon S-box implementation, we observe a reduction in the amount of cycles for encryption and decryption of approximately 30% when compared

to the portable implementations of [36]. For the PAEF-ForkSkinny-128-288 instance with three tweakey matrices, this speed-up is a bit lower (27%). This can be explained by the larger relative importance of the tweakey calculations in this instance. The ROM size is reduced by approximately 500 bytes in all 128-bit instances. This follows from the smaller code size of the round function, which now uses the parallel Neon S-box implementation. The amount of RAM needed for encryption or decryption remains the same.

Table 3: Implementation figures for the Neon SIMD implementations of ForkAE on the ZYBO platform.

	Encryption			**Decryption**		
	c/B	ROM	RAM	c/B	ROM	RAM
PAEF-FS-64-192	1184	3235	331	1390	2653	392
PAEF-FS-128-192	736	2619	161	807	2551	810
PAEF-FS-128-256	737	2651	169	806	2583	818
PAEF-FS-128-288	1026	2863	189	1078	2783	950
SAEF-FS-128-192	743	2491	161	812	2415	810
SAEF-FS-128-256	743	2519	169	810	2419	818

The execution time of PAEF-ForkSkinny-64-192 encryption improves with almost 500 cycles per byte, for example 29%, when compared to the portable implementation from [36]. For decryption, the speed-up is smaller as the degree of parallelism is lower. With the Neon inverse S-box implementation, we still accelerate decryption with 17%.

A single round of the 64-bit SKINNY round function with a Neon S-box implementation executes in 95 clock cycles on the ARM Cortex-A9. With 17 rounds before the forking point and 23 rounds after the forking point, a single branch of the ForkSkinny primitive, which is equal to the execution of the SKINNY primitive, needs 40 of these rounds. Producing twice as much output by calculating both branches requires $17 + 2 \times 23 = 63$ such rounds, or $\frac{63}{40} = 1.58$ times the amount of computations. When the S-box is calculated in parallel for the branches after the forking point, two rounds are calculated in 112 clock cycles instead of 2×95. This way producing the double output requires only 1.10 times the amount of execution time of a single branch.

Other SKINNY-based candidates need $M+1$ calls to the SKINNY primitive for M message blocks, while ForkAE, which has no fixed cost, needs M calls to the ForkSkinny primitive (with 1.10 times the computational cost). As a result, for implementations where no mode parallelism is exploited (for serial modes, like Romulus), ForkAE encryption will be faster for messages of up to 10 blocks. This is illustrated in Figure 7. Note that the SKINNY-AEAD and Romulus submissions to the NIST LWC competition do not include an instance with 64-bit blocks. However, when 256-bit SIMD hardware is available, this result could also be extended to the 128-bit instances.

5 Conclusion and future work

ForkAE is an efficient candidate for its core targeted setting of applications with short messages, however in the light of the recent results, it is also emerging as a suitable design for efficient (authenticated) encryption for longer messages. ForkAE has also been shown to give very strong security guarantees and can rightfully be placed in a category of efficient lightweight schemes providing 'defense in depth'. As illustrated, ForkAE with its SAEF

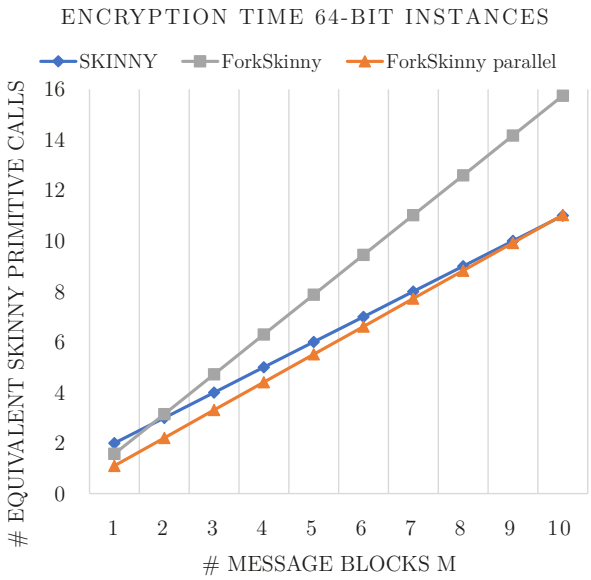


Fig. 7: Comparison of encryption time of 64-bit ForkAE implementations with SKINNY-AEAD, expressed in number of equivalent calls to SKINNY primitive.

mode achieves security even in the face of attackers exploiting nonce repetitions and/or being able to adaptively change the chosen plaintext block-by-block, which pose a very realistic attack vector in the lightweight setting due to the frequently constrained nature of the used HW platforms, frequent exposure to powerful attacks,or due to the distributed and hence 'unsupervised' nature of some of the applications. Additionally, PAEF and RPAEF achieve full n-bit security relevant for example in use cases where the key will never be refreshed and life-long devices with frequent utilization span is expected, or when the use of the most lightweight primitives (with 64-bit block) is desirable. Additionally, we show that the ForkSkinny primitive also lends itself well to the 'defense in depth' design aspect when the ForkSkinny primitive is used in an counter-style encryption.

ForkAE is flexible to be used in various modes, covering the needs of both fast and robust encryption. An interesting avenue is to explore the improvements offered by the ForkSkinny when used in an authentication-only mode. Our preliminary observations indicate both robust beyond birthday bound security and efficiency advantages.

References

[1] 3GPP TS 22.261: Service requirements for next generation new services and markets. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3107

[2] 3GPP TS 36.213: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2427

[3] NB-IoT: Enabling New Business Opportunities. http://www.huawei.com/minisite/iot/img/nb_iot_whitepaper_en.pdf

[4] Specification of Secure Onboard Communication. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf

[5] Detop dexterous transradial osseointegrated prosthesis with neural control and sensory feedback. http://www.detop-project.eu/ (2016)

[6] Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., Yasuda, K.: COLM v1 (2014), "https://competitions.cr.yp.to/round3/colmv1.pdf"

[7] Andreeva, E., Bhati, A.S., Preneel, B., aDamian Vizár: 1, 2, 3, Fork: Counter Mode Variants based on a Generalized Forkcipher. in submission to a peer-reviewed conference

[8] Andreeva, E., Bhati, A.S., Vizár, D.: Nonce-Reuse Security of the Online SAEF ForkAE mode. in submission to a peer-reviewed conference

[9] Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkae v.1
[10] Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: A new primitive for authenticated encryption of very short messages. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11922, pp. 153–182. Springer (2019). https://doi.org/10.1007/978-3-030-34621-8_6, https://doi.org/10.1007/978-3-030-34621-8_6
[11] Banik, S., Bossert, J., Jana, A., List, E., Lucks, S., Meier, W., Rahman, M., Saha, D., Sasaki, Y.: Cryptanalysis of forkaes. Cryptology ePrint Archive, Report 2019/289 (2019), https://eprint.iacr.org/2019/289
[12] Bariant, A., David, N., Leurent, G.: Cryptanalysis of forkciphers. IACR Transactions on Symmetric Cryptology **2020**, 233–265 (May 2020), https://tosc.iacr.org/index.php/ToSC/article/view/8564
[13] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-Hash
[14] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. pp. 123–153 (2016). https://doi.org/10.1007/978-3-662-53008-5_5
[15] Böck, H., Zauner, A., Devlin, S., Somorovsky, J., Jovanovic, P.: Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS. In: 10th USENIX Workshop on Offensive Technologies (2016)
[16] Deprez, A.: Optimized Software Implementations for the Lightweight Authenticated Encryption Schemes. Master Thesis, KU Leuven, 2020
[17] Deprez, A., Andreeva, E., Mera, J.M.B., Karmakar, A., Purnal, A.: Optimized Software Implementations for the Lightweight Encryption Scheme ForkAE. to appear in CARDIS 2020
[18] Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In: Canteaut, A. (ed.) Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. Lecture Notes in Computer Science, vol. 7549, pp. 196–215. Springer (2012). https://doi.org/10.1007/978-3-642-34047-5_12, https://doi.org/10.1007/978-3-642-34047-5_12
[19] Fouque, P.A., Joux, A., Martinet, G., Valette, F.: Authenticated On-Line Encryption. In: Matsui, M., Zuccherato, R.J. (eds.) Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003. Lecture Notes in Computer Science, vol. 3006, pp. 145–159. Springer, Ottawa, Canada (2004). https://doi.org/10.1007/978-3-540-24654-1_11, https://hal.inria.fr/inria-00563967
[20] Guillaume Endignoux, D.V.: Linking Online Misuse-Resistant Authenticated Encryption and Blockwise Attack Models. Cryptology ePrint Archive, Report 2017/184 (2017), https://eprint.iacr.org/2017/184
[21] Hoang, V.T., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 15–44. Springer (2015)
[22] Hoang, V.T., Reyhanitabar, R., Rogaway, P., Vizar, D.: Online Authenticated-Encryption and its Nonce-Reuse Misuse-Resistance. vol. 9215, pp. 493–517. Gennaro, R, Springer Verlag (2015)
[23] Liskov, M.D., Rivest, R.L., Wagner, D.A.: Tweakable block ciphers. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 31–46. Springer (2002). https://doi.org/10.1007/3-540-45708-9_3, https://doi.org/10.1007/3-540-45708-9_3
[24] O'Donnell, L.: "2 Million IoT Devices Vulnerable to Complete Takeover". Threatpost (2019), "https://threatpost.com/iot-devices-vulnerable-takeover/144167/"
[25] O'Donnell, L.: "Serious Security Flaws Found in Children's Connected Toys". Threatpost (2019), "https://threatpost.com/serious-security-flaws-found-in-childrens-connected-toys/151020/"
[26] Peyrin, T., Seurin, Y.: Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 33–63. Springer (2016). https://doi.org/10.1007/978-3-662-53018-4_2, https://doi.org/10.1007/978-3-662-53018-4_2
[27] Pittevils, J.: Low-area Optimized Hardware Implementations for ForkAE. Master Thesis, KU Leuven, 2020
[28] PLC, M.: Wireless tyre pressure monitoring (wtpms). https://www.meggitt.com/products-services/tyre-pressure-monitoring/

[29] Purnal, A., Andreeva, E., Roy, A., Vizár, D.: What the Fork: Implementation Aspects of a Forkcipher. In: NIST Lightweight Cryptography Workshop 2019 (2019)
[30] Rogaway, P.: Authenticated-encryption with associated-data. In: Atluri, V. (ed.) Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002. pp. 98–107. ACM (2002). https://doi.org/10.1145/586110.586125, https://doi.org/10.1145/586110.586125
[31] Rogaway, P., Shrimpton, T.: A provable-security treatment of the key-wrap problem. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 373–390. Springer (2006)
[32] Semiconductor, N.: nrf52840 product specification v1.1 (2019), https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf
[33] Standard, O.: Mqtt version 5.0. Retrieved June **22**, 2020 (2019)
[34] Vanhoef, M., Piessens, F.: Key reinstallation attacks: Forcing nonce reuse in WPA2. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1313–1328. ACM (2017)
[35] Vaudenay, S., Vizár, D.: Can caesar beat galois? - robustness of CAESAR candidates against nonce reusing and high data complexity attacks. In: Preneel, B., Vercauteren, F. (eds.) Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10892, pp. 476–494. Springer (2018). https://doi.org/10.1007/978-3-319-93387-0_25, https://doi.org/10.1007/978-3-319-93387-0_25
[36] Weatherley, R.: NIST lightweight cryptography primitives. GitHub repository (2020), https://github.com/rweather/lightweight-crypto